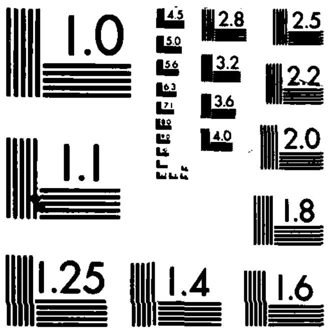
AD-A165 511 PRELIMINARY DESIGN CONCEPTS FOR COMMAND AND CONTROL 1/1
MODELING USING TIME W... (U) MITRE CORP MCLEAN VA MITRE
C3I DIV S J LASKOWSKI ET AL. 14 AUG 85 MTR-84W00535
UNCLASSIFIED F19628-84-C-0001 F/G 15/3 NL



MICROCOPY RESOLUTION TEST CHART

DARDS-1963-A

(1)

MITRE

WORKING PAPER

The MITRE Corporation
MITRE C³I Division
Washington C³I Operations
1820 Dolley Madison Boulevard
McLean, Virginia 22102

WP- 84W00535 No. Vol. Series Rev. Supp. Corr.

MITRE-Sponsored Research Program

AD-A165 511

Subject: Preliminary Design Concepts For Command and Control Modeling Using Time Warp/Hypercube

To: R. Peter Bonasso

From: S. J. Laskowski, R. O. Nugent, L. M. Sokol

Contract No.: F19628-84-C-0001

Dept.: W74

Sponsor: Army Model Improvement Program

Date: 14 August 1985

Project No.: 86080

Approved for MITRE Distribution: R. P. Bonasso

ABSTRACT:

MITRE was tasked by the Army Model Improvement Program Management Office to investigate design concepts for modeling command and control on the Time Warp/hypercube software/hardware combination. This working paper describes the objectives, considerations, concepts, evaluation methodology, and evaluation summary of this investigation.

86 3 13 041

~~85 12 9 169~~

ACKNOWLEDGMENTS

The authors express their gratitude to Dr. David Jefferson, University of California, Los Angeles; Dr. Philip Klahr, the RAND Corporation; and Henry Sowizral for their willingness to answer questions and discuss various details of object-oriented simulation, parallel processing and the time warp operating system. Within MITRE, the authors would like to acknowledge the contributions of Jim Antonisse, Ari Zymelman, and Ray Wong for their comments and suggestions in regard to the use of the time warp/hypercube for command and control modeling in combat simulations.

A-1

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

This report presents the results of MITRE Project 86080. The objective of this task was to develop and evaluate preliminary design concepts for modeling command and control (C^2) on the hypercube parallel processing computer architecture using the associated Time Warp operating system. MITRE performed this task in support of the Army Model Improvement Program (AMIP) Management Office (AMMO) systems research and planning efforts required as part of the development of a new family of Army models.

Command and control can be thought of as a large complex system of facilities, equipment, communications, procedures, and personnel which provide the means through which the Secretary of the Army, Navy, or Air Force; the Chiefs of Staff; and Commanders in the chain-of-command exercise command and control of forces and resources in performing the missions and functions assigned to them. Modeling C^2 provides a means of analyzing the process and the effects of alternative doctrine, tactics, and C^2 systems. The size and complexity of the command and control decision process make it difficult to model; simulation is one means of making the modeling problem tractable. → next page

Certain types of simulations provide a computational means of exploring complex, irregular, discrete systems that are difficult to analyze by other methods. Although simulation is a powerful tool, computational restrictions often mean that a simulation can take days of computer time to simulate an hour of real time. For example, one C^2 simulation requires two to six hours of computer time to simulate one hour of real time. Not only a decrease in the ratio of simulation time to real-world time, but also a more accurate reflection of the real world is desirable for applications to C^2 problems. However, speed and accuracy are typically conflicting objectives; for example, an increase in information may result in both an increase in accuracy and a decrease in simulation speed.

More faithful and faster C^2 simulations are required for the efficient and effective evaluation of emerging C^2 systems as well as doctrine. Object-

oriented programming [12] is one technique that has proved useful in building better C^2 simulations and, hence, achieving better C^2 representations. (For further information on object-oriented programming, see Section 2.1.) However, advances in large-scale simulations such as object-oriented simulations (and symbolic programs, in general) have been constrained by the serial nature of current traditional computer architectures. Single-processor architectures limit a simulation because each simulation event must take its turn executing on the processor. The advent of parallel computer architectures has opened up a whole new area of investigation into improving the execution time of these simulations.

Existing C^2 models have to work around trying to simulate parallel, independent real-world events. The use of parallel computing has the advantage that it would no longer require the transformation of parallel event representations into a sequential time sequence. The hope is that distributing discrete portions of a single simulation among many processors in parallel will speed up execution time by magnitudes of 10 to 100.

1.1 Objective

The objective of this effort was to ~~assist AMMO by~~ develop~~ing~~ and evaluat~~ing~~e design concepts for modeling command and control on the hypercube parallel processing computer architecture using the associated Time Warp operating system. In particular, the evaluation was to be responsive to two basic questions:

- o Can Time Warp on a hypercube architecture be used in conjunction with object-oriented techniques to significantly speed up the processing time associated with command and control modeling?

- o What are the resource implications (memory, communications, input/output) for the use of a Time Warp and hypercube mechanism?

1.2 Approach

The key to achieving an efficient implementation of a simulation on a hypercube is the careful mapping of the simulation to the processors. A poor

placement of a C^2 simulation on the hypercube could result in worse than serial execution time; on the other hand, an optimal placement strategy is not obvious. Our approach to finding this optimal strategy can be described by three related tasks. The first task finding was to evaluate whether Time Warp and hypercube could be integrated with an object-oriented simulation, a strategy which assigned portions of the simulation to the different processors to take advantage of the parallel processing. The second task was to develop a means of evaluating alternative assignment strategies so that one strategy can be judged better than another strategy. The final task was to develop an assignment optimization strategy to take advantage of the gains of parallel processing.

1.3 Organization

The remainder of this report is organized as follows. Section 2.0 contains background material on object-oriented programming, hypercube, and Time Warp. Section 3.0 contains the design considerations for the placement of an object-oriented program on a hypercube. Section 4.0 details the design decisions involved with the assignment of the simulation to hypercube processors and an evaluation strategy which can be used to evaluate alternative assignment strategies. Section 5.0 presents our findings and observations, while the study's conclusions are presented in Section 6.0. The Appendix contains a briefing presented to AMMO outlining both the project and MITRE's conclusions.

THIS PAGE INTENTIONALLY LEFT BLANK

2.0 BACKGROUND

The limited representation of C^2 in existing combat simulations is one of several concerns that prompted the establishment of AMMO, and it continues to be a concern of the AMMO office. AMMO's charter is to develop a comprehensive and well-integrated Army Modeling capability at the earliest date. MITRE previously has supported AMMO in its development of the functional area representative objectives for the Corps and Division Evaluation Model (CORDIVEM). These were developed utilizing work previously done on the functional segment subsystem specifications. At the same time, MITRE conducted a technical assessment of selected Army simulation models as candidates for the automated CORDIVEM [8]. Still another task conducted in support of the AMMO office was the evaluation of object-oriented programming for use in Army simulation modeling [9].

As part of this last evaluation, MITRE used the RAND Corporation's Rule Oriented Simulation System to develop a ground combat simulation entitled the Battlefield Environment Model (BEM) to examine the application of object-oriented programming. This MITRE evaluation showed that the benefits associated with the object-oriented approach to simulations include increased modifiability and intelligibility and improved performance characteristics; however, this was at a cost of large computer memory and slow running time. Parallel processing is regarded as a possible means of getting acceptable running times while keeping the increased representation.

Use of the hypercube architecture will require the placing of an object-oriented simulation on the multiprocessors; this requires the breaking of the simulation into manageable pieces, the assignment of these pieces to processors, and the synchronization of the various simulation pieces as they will be running at different simulation speeds. This is the problem that the Time Warp operating system is designed to handle, the time synchronization problem. The following sections will discuss object-oriented programming, hypercube and Time Warp.

2.1 Object-Oriented Programming

Object-oriented simulations are centered around objects (actors in the simulation) which can communicate only by passing messages between objects and are distinguished by having an action or a behavior in the simulation initiated by the receipt of a message sent by another object. Objects in this type of simulation have properties and behaviors which represent the characteristics, activities, and capabilities of the objects being simulated. The receipt of messages triggers behaviors in these objects akin to real-world objects receiving communications and causes the objects to perform certain actions. The objects are created in a hierarchy which permits inheritance of properties and behaviors; e.g., a generic Army battalion can be created with properties such as speed and number of trucks. This generic battalion then can be replicated with all its properties and each replication can inherit these properties from the generic battalion. In summary, the primary characteristics of object-oriented programming are the use of objects, message passing characteristics, and inheritance of properties and behaviors.

2.1.1 Objects

There are two types of objects used in object-oriented simulations: basic and auxiliary objects. Basic objects are the actors in the simulation in that they have some real-world representation such as the military units being simulated. There can be a one-to-one correspondence between real military objects and basic simulation objects, e.g., an Army battalion and simulation object representing a battalion, or possibly military objects can be represented by more than one type of simulation object. The basic simulation objects must have characteristics and behaviors sufficiently defined for them to portray adequately the system or units being represented. All action in the simulation is controlled through the passing of messages. Simulation object behaviors are in the form of IF-THEN rules which list the actions to be taken by an object on receipt of particular messages.

The second type of simulation object, an auxiliary object, has no real-world counterpart, but is used for support and control of the simulation.

Auxiliary objects can be used to support the simulation in various ways but are not directly involved in the simulation. Auxiliary objects can assist in the construction and conduct of the simulation; for example, an auxiliary object might be used to centralize most of the mathematical computation, thus making the basic object behaviors cleaner and therefore more readable. Auxiliary objects are also used to centralize common procedures such as movement calculations and can be used to maintain databases, such as a time-dependent terrain database to which simulation objects need access. For example, an actor called Terrain would be used because the overhead involved with keeping the entire terrain database on each object's property list is undesirable.

2.1.2 Message Passing

Simulation objects pass messages between themselves which activate object behaviors. There are two types of messages in the simulation: those which simulate real-world communications and those which might be called system messages. This first type of message triggers actor behaviors which simulate real military unit reactions to messages. It is this combination of message passing of the first type and rule-based behavior which provide a close parallel to the communications and the command and control which are necessary for a credible combat simulation. The latter message type includes those messages which might be used for simulation control or for data access.

2.1.3 Simulation Time

Because all action in the simulation takes place through behaviors activated by simulation messages, the object-oriented simulation can easily be executed in serial by executing the simulation messages in time order. The BEM is currently executed in a time-stepped, sequential manner, e.g., all actions scheduled for simulation time 5 are executed before simulation time 6. Sequential execution prevents time anomalies which might occur in parallel execution. Regardless of the type of execution, a time history of all (or selected) messages provides a trace of cause and effect in the simulation. We shall see later that time plays an especially important role when the

object-oriented simulation is transformed from its current sequential form into a form appropriate to parallel architecture.

2.1.4 Inheritance Hierarchy

In object-oriented programming, specific instances of objects can be created from generic classes of objects, forming a hierarchy of objects. Descendants in a hierarchy can inherit the properties and behaviors of hierarchical ancestors; just as a child inherits characteristics from both parents. For another example, a generic simulation unit, "Tank Co," could inherit properties and behaviors from a higher simulation unit called "Action Unit" (Figure 1). The advantage of this inheritance principle is that common properties and behaviors need to be entered only once in the construction of the simulation. This simplifies object modification and reduces the amount of storage required.

2.2 Hypercube

In pursuit of this challenge to obtain faster and more accurate simulations, the U.S. Army is developing a prototype hardware/software system based on a "hypercube" processor[14]. The hypercube is a multiprocessor architecture designed at California Institute of Technology and built at Jet Propulsion Lab. The primary advantage of a hypercube is that 2^n processors can work in concert on a particular problem and hopefully solve the problem faster than one computer. This does not mean, however, that if n processors are working on a problem, it will be solved n times faster, Furthermore, it should be noted that adding processors can mimic a diminishing returns curve; at some point extra processors detract from the system.

An n-dimensional hypercube is constructed from 2^n nodes, each containing an identical, independent processor and local memory, all connected by high-speed links arranged in the topology of an n-dimensional Boolean hypercube. Thus a 3-D hypercube (shown in Figure 2) has 8 nodes

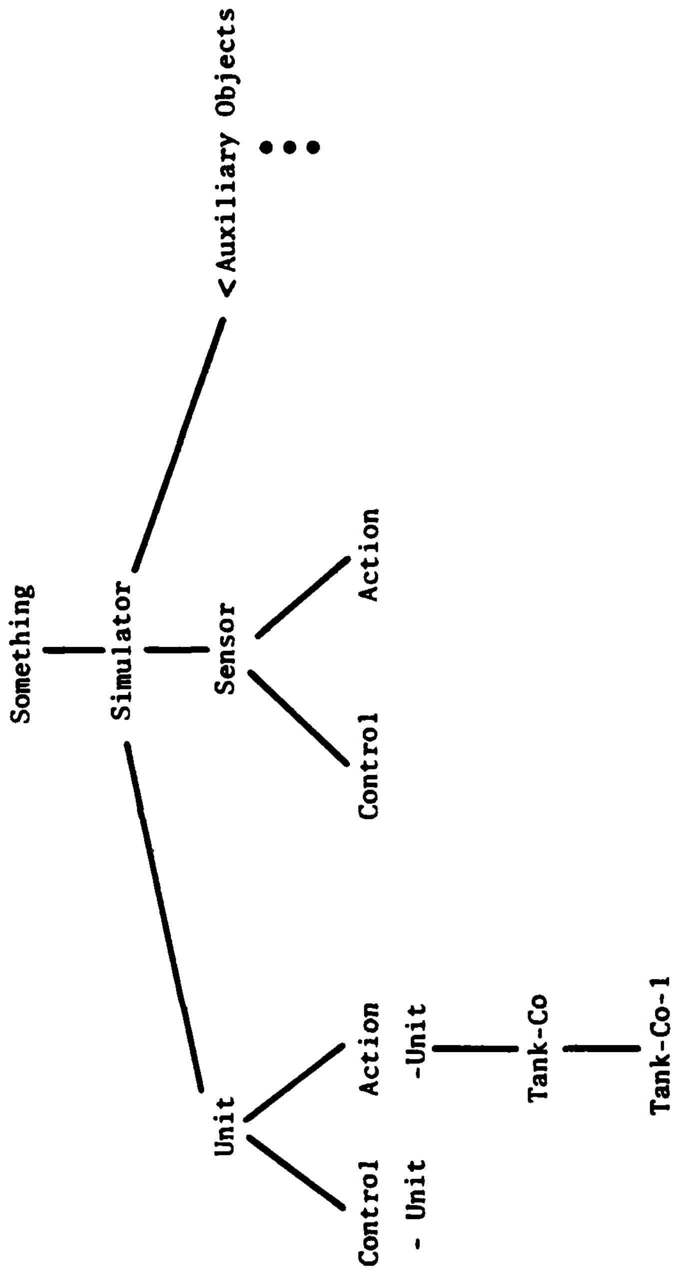


FIGURE 1
OBJECT HIERARCHY

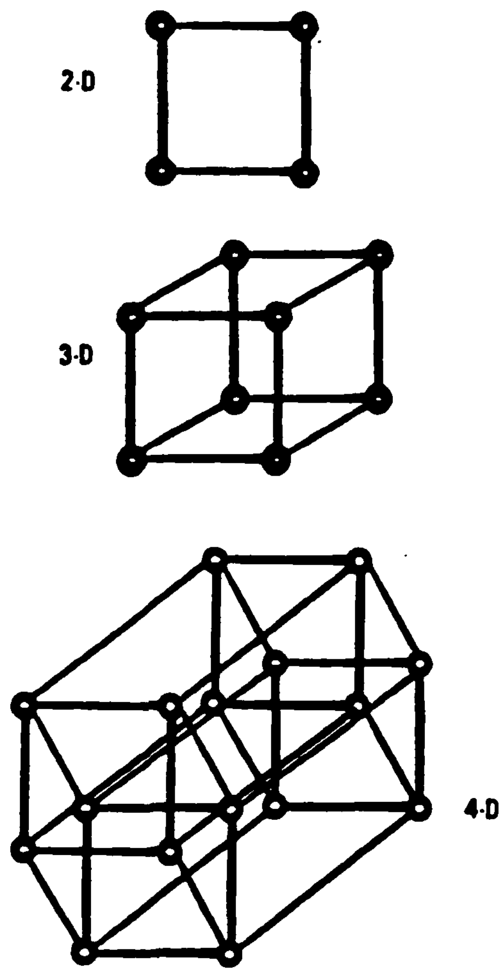


FIGURE 2
HYPERCUBES

connected in the pattern of an ordinary cube, and a 4-D hypercube has 16 nodes. Each node has exactly n nearest neighbors in this topology; the average distance between two nodes is n/2 and the maximum distance is n "hops" (where a hop is the distance as measured by processors, between any two nodes). The current generation of hypercubes has no shared memory. Processors may share information through message passing, but there is no central memory or database.

Many different parallel processing architectures are under development, each with its own unique set of characteristics (e.g., shared memory) and communication network. The hypercube can be considered a middle-of-the-road architecture with respect to two criteria: the average number of hops required for communication and the number of processor communication links. One extreme of the multiprocessor architecture spectrum is where each processor is connected to two other processors, for example, a ring. This architecture's disadvantage is that in order for two processors to communicate, they may have to go through many other processors and incur large communications overhead. An example of the other end of the architecture spectrum is where each processor is connected to every other processor. The number of processors allowed in most architectures is limited by the number of input/output ports on each processor, typically 15 for the hypercube.

2.3 Time Warp

In conjunction with the development of hypercube, the Army is also supporting the development of the Time Warp operating system, a specialized operating system for distributed simulations. Time Warp is a mechanism invented at the RAND Corporation in 1981 which allows the objects to proceed asynchronously through simulation time; this means that some of the objects are allowed to progress ahead in simulation time while others lag behind. Time Warp deals with the time anomalies caused by the asynchronous processing.

The Time Warp mechanism allows an object to go forward in simulation time until it receives an event message that it should have executed in its simulated past. An object can receive messages in its past because each processor can be at a different simulation time. For example, suppose a military battalion object's simulation time is 0235 hours while the division headquarters' simulation time is 0130, and that the division headquarters directs the battalion to move now. Division headquarters "now" is 0130; this time is in the battalion's past. The battalion must therefore rollback to a simulation time earlier than 0130 and start re-executing its part of the simulation (Figure 3); the object is able to roll back to an earlier time because it has saved previous messages and states. The insertion of the new message may change an object's future, and as a result its old set of future states may no longer be valid. However, this implies that some of the messages the object has sent out may also be invalid. Therefore, as the object goes forward in simulation time, it must check to see if it has sent out any inappropriate messages or caused any inappropriate side effects. To correct this problem, as the object goes forward in simulation time, it checks each message's validity. If the object does find an invalid message, it sends an anti-message to annihilate it; any messages which remain valid are left unchanged. The receipt of an anti-message can cause a secondary rollback. The Time Warp however does guarantee a forward progression in time and that the process is free of deadlock[11]. The main feature distinguishing it from other distributed simulation mechanisms (e.g., those for scientific applications) is its reliance on distributed rollback and "anti-messages" for synchronization and flow control[11].

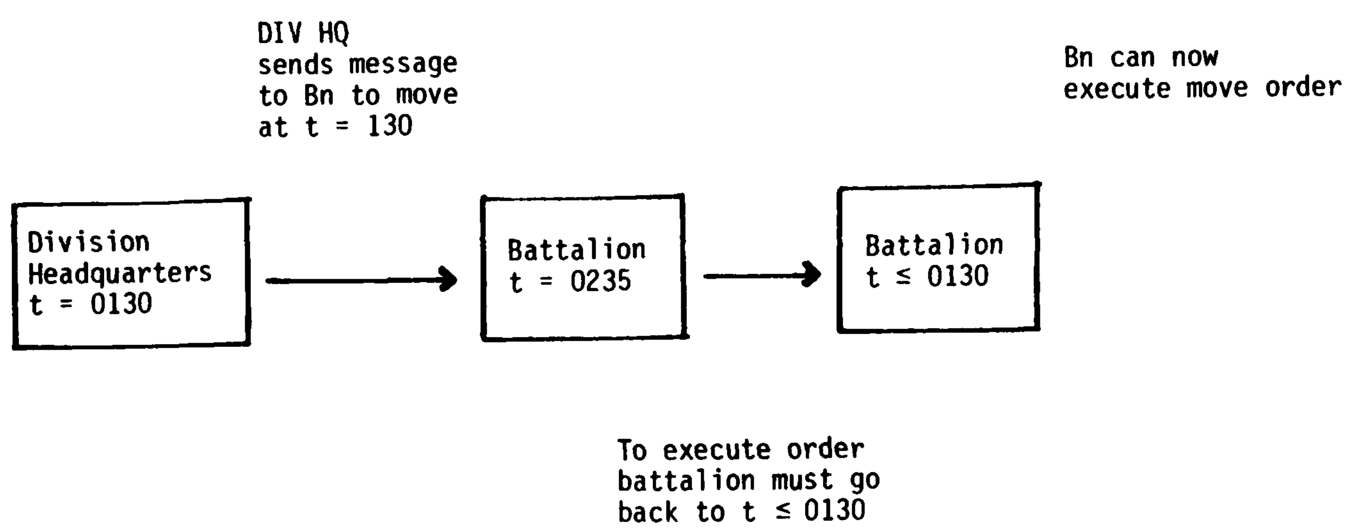


FIGURE 3

ROLLBACK FLOWCHART

THIS PAGE INTENTIONALLY LEFT BLANK

3.0 PORTING CONSIDERATIONS

A C^2 object-oriented simulation consists of a collection of simulation objects representing military units with real behaviors, procedures, and large common databases. The simulation objects are used to mimic the actions of military units, e.g., communicating messages, processing information, changing location, etc. Simulation objects are used to represent real military units performing (sequential) processes that can executed in parallel and can communicate with each other by exchanging messages. In Time Warp, these messages are time stamped with a simulation time, and the order of event execution is determined by the time-stamp on a message.

Real events are events which occur in parallel in the physical world and are independent of one another and can be simulated concurrently. This does not mean that a simulation with 100 objects on 100 processors can be made to go 100 times faster through the use of parallel architectures; it does mean that the degree of achievable computational concurrency is directly proportional to the degree of physical concurrency or independence in a model. This means that if the real events are both sequential and dependent, it is very difficult to separate events and execute them in parallel. This level of potential computational concurrency provides an upper bound to how much a simulation can potentially be speeded up; if there is little concurrency in a particular representation, its implementation on a parallel architecture will not greatly impact its execution time.

The porting of an object-oriented simulation onto a hypercube architecture requires that the simulation be placed on more than one processor. This means that the representation must be decomposed in such a way that its subsets can be assigned to different processors, more than one subset can be assigned to one processor. Note that a poor decomposition of the simulation onto the parallel processor can result in a worse than serial implementation.

Two strategies are needed to port an object-oriented, message-passing simulation onto a Time Warp/hypercube system: the assignment of objects to

processors and the assignment of information and procedures to objects. These assignment strategies are discussed in Sections 3.2 and 3.3.

3.1 Constraints

There are, of course, factors that impose constraints on any assignment strategy: the size of the processor memory, the processor speed, and the speed of communications. Memory size limits the number of objects on a processor and similarly limits the amount of information and procedures on each processor. The optimal number of processors for a particular implementation will also be a function of memory size; the smaller the memory, the greater the number of processors required. Processor speed will constrain the number of objects which can be assigned to a processor; if a slow processor is assigned many active units, it will proceed very slowly through simulation time. Slow machine communications will strongly influence the assignment of information to objects and the assignment of objects to processors; slow machine communication speed makes it cheaper to store information than to access it elsewhere. Furthermore, slow machine communication speeds will make the simulation slower. The optimization strategies associated with these constraints are discussed in Section 4.0.

3.2 Object Assignment Strategies

There are three basic strategies for assigning objects to processors. The first strategy is to place all objects on one processor; the result is slow serial implementation. The second strategy places each object on its own processor. A pure implementation of this strategy is limited by the number of processors available. The third strategy clusters objects on processors, i.e., places more than one object on a processor. The degree of this clustering is influenced by processor characteristics, simulation characteristics, and the number of messages passed between objects. Communications speed may make it desirable to place objects which communicate frequently within the same processor, or at least on processors close to each other, to reduce communications overhead.

3.3 Information and Behavior Assignment Strategies

Similarly, there are three basic strategies to assign information and behaviors to objects. The first strategy is to give each object all possible pertinent information, implying a high degree of replication and associated database synchronization overhead. The second strategy is to give each object most of its frequently used information. The object then would have to query other objects for information that it does not have. This means that there is a lower degree of information replication throughout the system. The advantage of this is that synchronization costs are lower; the disadvantage is that an object will have to spend more time acquiring information. The last strategy assigns each object a minimum of information. This strategy implies very little information replication and, therefore, low database synchronization costs. It does however mean that each time a unit needs a piece of information that it must go acquire it; this strategy would have high waiting time costs.

3.4 Expected Bottlenecks

There appear to be two possible sources of major bottlenecks to achieving satisfactory performance results after problem decomposition and object assignment are dealt with. The first of these possible bottlenecks is the sequential nature of many command and control processes, at least at the higher echelons of command. A high degree of centralized control also would make the top level control unit in the simulation a possible bottleneck. The second area in which bottlenecks may be created is that of large databases to which many objects need access and which may be time dependent. Methods for dealing with database bottlenecks are discussed in Section 4.4.

THIS PAGE INTENTIONALLY LEFT BLANK

4.0 IMPLEMENTATION STRATEGY AND EVALUATION

The performance of a model on a sequential computer is usually measured by the number of computer instructions (and/or operations) required to complete the simulation. This type of standard analysis is insufficient for parallel implementations because of all the additional type operations present, e.g., communication and synchronization. Therefore, this standard measure cannot be used to gauge the decisions which must be made relative to the assignment of objects to processors and the assignment of information and procedures to each object.

Therefore, an alternate evaluation scheme is needed to compare alternative assignment strategies; one scheme is to use total execution time as a global evaluation measure. Execution time of a simulation running on multiple cooperating processors can be defined as the time elapsed from when the first processor begins until when the last processor finishes. If the execution time on any given processor is defined as the sum of the busy and idle times for that processor, execution time will be the same for all processors. The implementation strategy should attempt to minimize the execution time for all processors.

4.1 Assumptions

Two initial assumptions are made to facilitate the initial development of the evaluation measure; we will discuss the ramifications associated with the loosening of these assumptions later in the paper. The first assumption is that sufficient processor memory is available. This implies that, at least initially, memory limitations are not an overriding concern. This is a reasonable assumption to make, as the currently available hypercubes (e.g., Intel and Mark II) have a reasonable amount of memory (between .5 and 4 megabytes) on each processor. This assumption allowed us to hypothesize what information and object assignment strategies make intuitive sense, without being bound by memory constraints.

The second assumption is merely that communication speed is reasonable relative to this application. Slow communication speed would strongly influence the assignment of information to objects and objects to processors, as it makes the price of accessing information higher than storing it. At the very least, slow machine communication speed will slow down the simulation and increase the time difference between sender and receiver of messages.

4.2 Global Evaluation Measure

Execution time or E is used as the global execution time, which can be broken down into a four-part sum, consisting of P, processing time; W, wait or idle time; S, synchronization time; and C, communication time, i.e.,

$$E = P + W + S + C$$

Execution time begins when the first processor begins executing and ends when the last processor finishes execution. Execution time is the same for all processors, however the components of execution time will vary by processor.

Processing time reflects all the time spent by a processor processing information from one or more objects. Synchronization time consists of time a processor spends as part of the synchronization process (rollbacks and time updates). Wait time for each processor consists of the time spent idle as well as the time spent waiting for replies to queries. Communication time reflects time the processor spends communicating values from one processor to another (transmission time only)[1].

The communication component consists of three types of communications: query, update, and action messages. A query occurs when an object requests information from another object. An object can respond to queries without rolling back because it keeps a record of old state information. The amount of historical information which can be kept is partially a function of processor memory size. An update is a change made to a database and can cause rollbacks. An action message causes an object to do

something and can cause rollbacks. Rollbacks occur when either an update or an action message arrives at a processor with a time stamp earlier than the processor's local simulation time. These components will be referred to later as the query-communication component, the update-communication component, and the rollback-communication component.

Since each processor within the hypercube is actually a serial processor, the model assumes that each processor is serial and can perform only one process at a time. Otherwise each processor could do more than one thing at a time. This would make the processor a type of parallel processor, which could then be split into several serial processors. The model also assumes no overlap of processing, communication, and synchronization times; this implies a processor can handle only one task at a time.

4.3 Assignment Strategy Based on Object Characterization

An initial strategy for assigning information and procedures was chosen based on the command and control functions performed by each object, and its ease of implementation. Each military object participates to some degree in a set of four C^2 functions: plan, coordinate, direct and control. One way of modeling the C^2 process is to look at the various tasks which must be carried out in support of these functions. These tasks include collect information; evaluate plan, capabilities, and requirements; develop decision and transmit; allocate forces; coordinate; control, monitor, and adjust. In order for the simulation objects to be able to carry out these tasks, the simulation must include time ordering of supporting subtasks, data bases and knowledge bases, and rules for processing information. For example, in a military simulation, a typical high-level military object, such as a corps headquarters, spends much of its time processing and fusing information in order to make plans and issue orders. To perform this function it needs a considerable amount of processing capabilities and a large database which is kept current by updates from lower echelons.

4.3.1 High-Level Object Characterization

The characterization of an object's functions can be used to select an initial information and procedure assignment strategy which appears to match an object's complexity and characterization. This initial strategy implies assigning an object all the information appropriate to its characterization; e.g., if it uses a great deal of information, assign it large databases. For example, at the upper end of the spectrum in a corps-level simulation are the corps and division headquarters. While these headquarters are few in number, they are complex in the level of processing capability to be represented. The database maintained by top-level control units in the simulation should be both detailed and aggregated status information, and this must be both in present and projected terms. The corps planning horizon is 72 hours, and data forecasts are required to support that planning. The time horizon and larger area of interest imply large databases to contain the required information for planning and decisionmaking. In keeping with the data requirements of objects at this level, initial strategy would be to give a division or corps headquarters simulation object all necessary information and procedures because they need a large amount of information and processing capability to fulfill their mission.

Given an initial information assignment strategy and characterization of an object, the hypothesized performance components could be analyzed for the implications this initial strategy has for the execution measure. For example, the processing component in this case would probably be high because this type of object primarily plans and processes information. The component associated with query communication would probably be moderate due to queries from low-level objects. The action-communication component is a function of the particular simulation and independent of the amount of stored information. The large amount of information associated with this strategy implies a high degree of information replication; therefore, the update communication overhead probably would be high.

The amount of rollback due to action messages is a function of the virtual time difference between the sender and the receiver of the action message. The frequency of action rollbacks is independent of the amount of stored information. No rollback is caused if the update is for the current or a future time. The number of rollbacks caused by updates may in some part be a function of the number of updates, but more importantly it is a function of the virtual time difference between sender and receiver of updates.

4.3.2 Optimization Strategy for High-Level Objects

An appropriate question is whether this initial, intuitive strategy can be improved relative to the execution time measure? In fact, the strategy probably can be improved; however, the optimization is not straightforward because of the complex inter-relationships of the execution measure components. High amounts of processing time for a particular node could mean that the local virtual time of that particular processor might lag behind that of other processors and thus cause frequent rollbacks. One obvious way of reducing this component would be to place the object onto more than one processor, a type of load balancing. This can be done by dividing the functions of the represented object and creating two or more simulation objects.

The high update component also could be reduced if the amount of stored information were decreased. However, this decrease would mean that object would now have to query other objects for information and thus spend more of its time communicating queries and waiting for responses.

Rollbacks can be kept low by minimizing the virtual time difference between sender and receiver of updates/actions. The virtual time difference is minimized when all the processors run at approximately the same simulation speed. This can be accomplished by partitioning objects and by placing the partitions on different processors.

4.3.3 Characterization of a Low-Level Object

A resolution object (i.e., the lowest level object being simulated) such as an infantry battalion is concerned with its present, or near-term, status as

well as the mission and plan which have been given it. It is also concerned with and requires information regarding its status and that of opposing forces. It spends most of its time executing orders, obtaining information from other objects, and sending information to higher echelons. The object's characteristics indicate the amount of time spent planning and processing information is relatively small. The characterization of an object can again be used to select an initial information and procedure assignment strategy which appears to match a resolution object's characterization, e.g., low-level military objects such as an infantry battalion receive minimal information and procedures because the processing of information is not a primary function.

Given this initial assignment strategy and the characterization of the object, one can hypothesize about the components of the evaluation measure. The processing component would be low because this type of object does little processing of information (because there is little synthesis of information from subordinate units) and its special needs can be filled by database or specialized objects (procedure objects which provide functions such as Mathematician). The query-communication component would be high because the object's basic function is to obtain information, such as enemy position, from other objects. The action-communication component is a function of the simulation only an independent of the information assignment strategy. The minimal database implies minimal information replication and update overhead. Rollbacks are a function of the time differential between sender and receiver of a communication; the farther behind the sender is in virtual time, the greater the magnitude of the rollback.

4.3.4 Optimization Strategy for Low-Level Objects

Again, an appropriate question is whether this initial, intuitive assignment strategy can be improved relative to the execution time measure? Similarly, the answer is in the affirmative. Low amounts of processing imply that an object can race ahead relative to simulation time; however, this may not always be a desirable situation. If a battalion object were far ahead of its division in local virtual time, all the orders issued by the

division would cause rollbacks in the battalion, thereby increasing synchronization overhead. To prevent frequent rollbacks, it may be desirable (although nonintuitive) to slow the object (processor) down by clustering several low-level objects on the processor.

The high query and wait time components can be reduced through judicious additions to the object's information and procedures; however, any increase in the size of the stored database will also increase the update-communication component. The rollback component can be kept low by minimizing the local virtual time difference between the sender and receiver of updates or actions. In order to keep all the processors running at approximately the same simulation speed, the low-level objects must be prevented from getting too far ahead of some of the larger objects; this can be accomplished by clustering several of the low-level objects on a processor.

4.4 Common Databases and Procedures

C^2 simulations have two other components which must be dealt with to take advantage of parallel processing: procedures and common databases. Procedures are collections of related function routines, e.g., movement or mathematical functions. The procedure calls are grouped together to simplify the behaviors of the military objects; however, it is feasible to decompose these objects, replicate the functions, and place copies of the replications with each object. Common databases are collections of related information. Both of these components can be handled through the use of auxiliary objects. However, auxiliary objects frequently become simulation bottlenecks because of the number of simulation objects trying to use functions or access databases of the auxiliary objects. Auxiliary objects should be avoided, if possible, or implemented very carefully to minimize the bottleneck. The problem becomes somewhat akin to the distributed database problem: how many times should a database be replicated so that bottlenecks are prevented and at the same time high communications overheads are not incurred.

Database objects pose a special problem since they are frequently time dependent and the frequency of change for the C^2 common databases varies

considerably; e.g., ground truth changes frequently while terrain changes infrequently. Databases with high frequency of change cannot be freely replicated, because it can be difficult to keep the replications consistent. The frequency of change also impacts the frequency of rollbacks. Objects usually do not possess the discrimination to determine whether a rollback affects them. If a change in a pertinent database is made, a rollback must be initiated.

The database optimization strategy is primarily a function of the frequency of the database changes. For databases with low frequency of change, some form of replication on several processors can prevent simulation bottlenecks. The form of the replication varies. For those databases with low frequency of change, it is possible to copy the databases and to assign copies of the database to the objects as needed. For databases with high frequency of change, the overhead associated with keeping many copies is prohibitive. An alternative strategy would be to judiciously partition the database and to place the partitions on different processors; this would decrease the wait required to access the database. Both strategies should attempt to minimize the local virtual time difference between the database and the objects which use it in order to minimize the number of rollbacks.

4.5 Tightening of Assumptions

If the assumptions made earlier do not in fact hold, one might use a different strategy and obtain different results. For example, if the processor memory is limited, it is not feasible to give each object its own large database. Similarly, a tight memory constraint limits the number of old state copies that could be saved, the degree of object clustering possible, and, therefore, the ease of rollback, the degree of load balancing possible, and the amount of information and procedures which could be stored. Furthermore, if the processor memory is restrictive and only a limited number of processors are available, the possible gains for parallelizing a large simulation are small.

If communication speed is a tight constraint, it more strongly influences the assignment of information to objects, and objects to processors, as it

makes the price of accessing information higher than storing it. Additionally, slow machine communication speed slows down the simulation and increases the time difference between sender and receiver of messages.

Tight memory and communication constraints can to some degree be compensated by information paging techniques, judicious assignment and collocation of entities, judicious handling of auxiliary objects, load balancing, and frequent updates of the global system time.

4.6 General Optimization Strategy

Maximal simulation speed can be obtained through an iterative reorganization of the simulation, where object decompositions are varied. This alternative implies changing the definition of simulation objects and the assignment of both objects and information to objects. This reassignment and change in decomposition strategy would modify the value of the execution measure components. This change in component values unfortunately does not have a straightforward optimization strategy associated with it because, when one component is decreased, one or two of the other components usually increase. The optimization strategy is therefore an iterative one of object redefinition, and reassignment of object and information.

THIS PAGE INTENTIONALLY LEFT BLANK

5.0 FINDINGS AND OBSERVATIONS

Our findings and observations are classified in three categories: performance, Time Warp/hypercube's proposed integration with object-oriented programming, and projected human resource requirements.

5.1 Performance

It seems clear that good performance on the Time Warp/hypercube combination requires good problem decomposition in order to exploit parallelism. Bad decomposition may in fact produce results less satisfactory than straight serial processing.

Secondly, good performance will depend on good assignment of objects, behaviors, and data among processors. This assignment is necessary to balance the load between processors while at the same time keeping the processors running at nearly the same virtual time. It is desirable to minimize both rollbacks and communications; this is possible in part by load balancing and in part by object assignment.

Because there are no clear rules of thumb to determine initial good problem decomposition or assignment of objects to processors, testing and iteration will be required. The RAND Corporation is currently investigating techniques and possible development of an expert system for determining object assignment to take advantage of parallel processing.

5.2 Integration with Object-Oriented Programming

The use of the Time Warp/hypercube mechanism provides a favorable environment for the use of object-oriented programming that is seen as desirable for good command and control representation in a simulation. The environment does not, however, force the use of some desirable object-oriented techniques, e.g., not all of the necessary support features are inherently available in a simulation that represents objects, but does not have the other characteristics of object-oriented simulation.

The object-oriented design of the hypercube architecture does not by itself force the decomposition of the problem to objects making a good

representation of the components of systems. It is the responsibility of the designer to cleverly decompose the problem into objects of interest for the purpose for which the simulation is applied. The behaviors must also be constructed to model the decisionmaking processes utilizing information which would be used in the decision process.

In the Time Warp/hypercube mechanism, message passing is enforced when the objects are on different processors and also when they are on the same processor. Because of the message passing, the ability to trace cause and effect via message flow is inherently capable of being implemented.

There are some features available in object-oriented programming which are not automatically available and must be designed into a support package if desired. Parallel processing negates much of the utility of the object hierarchy, not only in construction of the simulation but also in the inheritance of object properties and behaviors. The inheritance of properties from generic objects is made difficult when generic and instance objects are not collocated on the same processor; this is because the instance object must query the generic objects for its inherited behaviors and properties.

5.3 Human Resource Requirements

Because of the problem decomposition complexity and the need for good assignment of objects of processors, it appears initially that there will be heavy resource (personnel, effort, and time) requirements associated with the use of the Time Warp/hypercube mechanism. It is important to point out that such a cost should be expected, at least initially, with a methodology this new. Because there is no large base of experience in problem decomposition and object assignment problems of this type, considerable experimentation and trials may be required before successful results are attained.

In order to facilitate the use of the Time Warp/hypercube mechanism, we recommend that a good package of simulation tools be made available to assist the user in the construction, use, and modification of the object-oriented simulations. If the basic language of the object-oriented simulation is to remain, a front end needs to be provided for the less sophisticated user.

Furthermore, while it may not be obvious, the Time Warp/hypercube mechanism requires a dedicated system while operating; that is, the hardware can only process one simulation at a time.

THIS PAGE INTENTIONALLY LEFT BLANK

6.0 CONCLUSIONS

It is our opinion that the continued funding of Time Warp/hypercube is worthwhile and that it will contribute towards the effort to obtain improved representations of command and control within combat simulations. Although there are some potential restrictions, we do not see any clear reason why there cannot be significant gains in the representation provided.

There are two reasons which exist for limiting the results to be expected. The first is that of problem decomposition which must be accomplished to exploit inherent parallelism while at the same time attaining the credibility of representation and the goals of the analysis to which the simulation is to be applied. The second reason lies in the trade-off which must be made between the amounts of processing and communications due to the assignment of simulation objects to processors.

In order that maximum advantage might be taken of the Time Warp/hypercube mechanism, several areas of research need to be pursued. The first is how to "cleverly" decompose a large serial simulation for placement on a parallel architecture. The second is how to "smartly" assign objects to the processors within the parallel architecture. The third area is how to handle databases and functions.

In the near term, we believe that implementing a command and control model on the Time Warp/hypercube mechanism will require considerable effort for the reasons previously discussed.

MITRE

APPENDIX A

Preliminary Design Concepts on Command And

Control Modelling Using Time Warp/Hypercube

MITRE

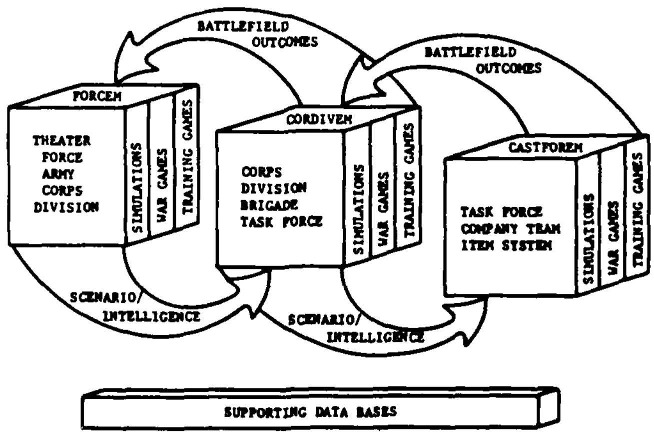
OUTLINE

- Objectives
- Problem Background
- Previous MITRE Support
- Definitions
- Command and Control Functions
- Implied Required Representation of Objects, Data, Procedures
- Design Alternatives
- Evaluation, Observation and Recommendations

MITRE

OBJECTIVES

- To develop preliminary design concepts for C^2 modelling on Time Warp/Hypercube (TW/HC)
- To evaluate
  - Performance
  - Integration with current architectures
  - Effect on use of object oriented techniques
  - Capability to deliver better C^2 representation
  - Resource requirements



MITRE

BACKGROUND

PROBLEM

- Army concern with limited C^2, logistics, etc representation in simulations
- Promises of object-oriented programming
- Limitations of object-oriented programming

POSSIBLE SOLUTION

- Parallel processing a way to overcome slowness and limited representation
- Time-warp looked at as a mechanism to speed up simulation by allowing processors to run asynchronously

MITRE

PREVIOUS MITRE SUPPORT

- Development of CORDIVEM functional area representation objectives
- Definition of functional segment subsystem specifications
- Technical assessment of Army simulation models
- Evaluation of object-oriented programming for Army simulation

MITRE

PREVIOUS MITRE SUPPORT (Cont'd)

- Use of ROSS in Battlefield Environment Model
- Functional Analysis of Command and Control Process
  - Maneuver Control

MITRE

DEFINITIONS

- Object Oriented Programming
- Parallel Processing
- Hypercube
- Time Warp

MITRE

OBJECT-ORIENTED PROGRAMMING

- Basic objects represent parts of system being studied
- Auxiliary objects may be used for computation or control
- Objects have properties and behaviors
- Messages between objects invoke behaviors
- Message passing aids tracing cause and effect

MITRE

OBJECT HIERARCHY

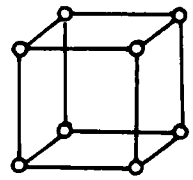
- Feature of some object-oriented programming languages/systems
- Allows storing of data and behaviors at highest common level (reduces storage)
- Aids in construction, modification and control of simulation
- Limited use on hypercube

MITRE

HARDWARE

HYPERCUBE

- Parallel processing
- 2^n Processors
- n Links
- Max n "Hops"
- All processors identical



MITRE

SOFTWARE CONTROL

TIME WARP/VIRTUAL TIME

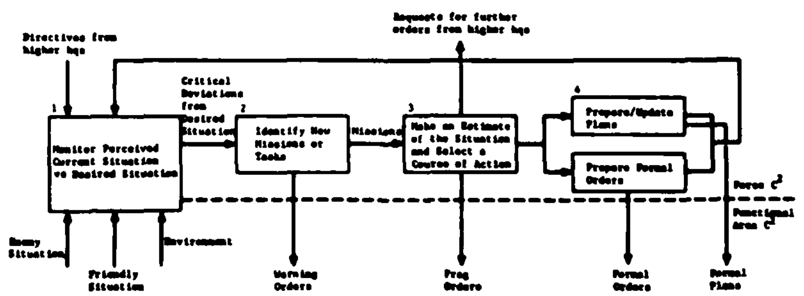
- Allows processors to run asynchronously
- Requires object-oriented, message passing architecture
- Messages queues and states periodically saved
- Object "rolls back" to a previously-saved state if necessary
- Global virtual time updated periodically

MITRE

C^2 PROCESS

Process Flow

- Monitor
- Detect
- Compare
- Decide



MITRE

C^2 TASKS

- Collection Information
- Evaluate plan, capabilities, requirements
- Develop Decision (& Transmit)
- Allocate Forces
- Coordinate
- Control - Monitor, Adjust

MITRE

EXAMPLE ELEMENTS OF INPUT INFORMATION

COMPARISON OF PERCEIVED AND
DESIRED SITUATION

- Commanders Guidance
- Friendly Mission
- Friendly Force Status
- Constraints

own info

- Enemy Force Status
- Key Terrain
- Weather Forecast
- Decision Criteria

outside info

MITRE

RESOLUTION OBJECTS

"ACTION" UNITS - BATTALIONS

'OWN' DATA

- Strength, Location, Status, Superior, Subordinate, Support, Adjacent, Mission, Plan

DATA BASES

- Terrain
- Weather
- Enemy Locations (perceived)
- Apriori knowledge of enemy, tactics, etc

KNOWLEDGE BASES

- Conditions under which to report, request, support, request relief, break contact, etc.

MITRE

TOP LEVEL OBJECTS

CONTROL UNITS - CORPS / DIVISION HEADQUARTERS

'OWN DATA' - HQ, aggregated, and front line BNS

- Strength, Location, Status, Superior, Subordinate, Support, Adjacent, Mission, Plans

DATA BASES (Present and Future)

- Terrain
- Weather
- Enemy locations (perceived, out to 72 hrs, 300 KMS)
- Present and forecast level of supplies and replacements
- Apriori knowledge of enemy, tactics, etc

KNOWLEDGE BASES

- Operation plan development and dissemination
- Information receipt and processing
- Comparison of perceived and desired situation
- Decision to modify/redo plan

MITRE

AUXILIARY ACTORS

- Handle computational load
- Make basic actor code cleaner
- Localize searches, interaction
- Serve as intermediary "controller"

MITRE

DESIGN AND EVALUATION - OUTLINE

- Design
- Evaluation
- Information and Procedure Allocation Strategies
- Optimization Strategy
- Auxiliary Objects
- Implementation Warning Signals
- Tightening of Assumptions

MITRE

C^2 DESIGN CONSIDERATIONS

- Simulation objects represent military units
- Other objects may be necessary to handle common procedures, database
- Functional variance of military objects
- Hierarchical relationship of objects
- Coupling of objects
- Amount of interaction between objects

MITRE

DESIGN VARIABLES

- Allocation of information and behaviors to processors
- Allocation of objects to processors

MITRE

ALLOCATION OF INFORMATION AND PROCEDURES TO OBJECTS

- Allocation Strategy - Object gets:
  - All necessary information and procedures
  - Some information and procedures
  - Its own information
- Constraints
  - Processor memory size
  - Communications speed and bandwidth
  - Processor speed

MITRE

ALLOCATION OF OBJECTS TO PROCESSORS

- Object Definition
- Allocation Strategies
  - All simulation objects on one processor
  - One simulation object on one processor
  - Several simulation objects on one processor (clustering)
- Constraints
  - Processor memory size
  - Communications speed and bandwidth
  - Number of processors available
  - Processor speed
  - Number of messages passed between objects

MITRE

INITIAL ASSUMPTIONS

- Sufficient processor memory available
- Fast communications relative to processing time

MITRE

EVALUATION

- Degrees of freedom exist in implementation
- How to evaluate alternative allocation strategies
- Evaluation based on total execution time

MITRE

EXECUTION TIME

- Execution time is defined as the time elapsed from when first processor begins to when the last processor finishes
- Execution time will be the same for all processors
- E = P + C + S + W

  Where:

  P = Processing time

  C = Communication time

  S = Synchronization time

  W = Idle or wait time
- Model assumes one serial process per processor; no overlap of processing, communication and synchronization times

MITRE

DEFINITIONS

- Three types of communication:
  - Query C_Q
  - Update C_U
  - Action C_A
- Two causes of rollback:
  - Update S_U
  - Action S_A

MITRE

INFORMATION AND PROCEDURE ALLOCATION STRATEGIES

Schema	MESSAGE FREQUENCY		E						
	Number of Queries	Number of Updates	P	C: C_Q	C: C_A	C: C_U	S: S_A	S: S_U	W
• All necessary information and procedures	L	H	H	L	N/A	H	N/A	?	L
• Some information and procedures	M	M	M	M	N/A	M	N/A	?	M
• Own information	H	L	L	H	N/A	L	N/A	?	H

H = High
M = Medium
L = Low
N/A = No impact

MITRE

VARIANCES IN OBJECT CHARACTERIZATION

- High Level Military Object, e.g., Corps HQ
  - Plans
  - Processes information
  - Issues orders
  - Receives information
- Resolution Military Object, e.g., Battalion
  - Gathers information
  - Receives orders
  - Sends information

MITRE

OPTIMIZATION STRATEGY

High Level Units

E	=	P	+	C_Q	+	C_A	+	C_U	+	S_A	+	S_U	+	W
		high		medium		N/A		high		?		?		medium

Problem Areas	Solutions
High P	- Allocate object to more than one processor (load balance)
High C_U	- Decrease amount of stored information - Decrease in C_U will increase C_Q and W
? S_A	- Minimize local virtual time difference between sender and receiver of action by load balancing
? S_U	- Minimize local virtual time difference between sender and receiver of update by load balancing

MITRE

OPTIMIZATION STRATEGY (CONTINUED)

Resolution Object

E	=	P	+	C_Q	+	C_A	+	C_U	+	S_A	+	S_U	+	W
		low		high		N/A		low		?		low		high

Problems Areas	Solutions
Low P	- Cluster low level objects (load balance)
High C_Q	- Increase amount of information in memory - Decrease in C_Q will increase C_U and P and decrease W
? S_A	- Minimize local virtual time difference between sender and receiver of actions by load balancing
High W	- Increase amount of information in memory - Decrease in W will increase C_U

MITRE

AUXILIARY OBJECTS

- Two types
  - Procedures
  - Common database
- Potential simulation bottleneck

MITRE

AUXILIARY OBJECTS

Procedures

- Not time based
- Provide computational functions, e.g., movement
- Decomposition possible
- Allocation Strategy - Optimization
  - Replication
  - Placement with objects

MITRE

AUXILIARY OBJECTS

COMMON DATABASE

- Time dependency
- Frequency of change varies greatly
- Frequent change can cause major rollback problems as military objects can't discriminate which information impacts them

MITRE

COMMON DATABASE AUXILIARY OBJECTS

ALLOCATION STRATEGY

- For databases with low frequency of change; e.g., terrain
  - Replicate
  - Page to objects
  - Minimize local virtual time difference between database and objects which use it
- For databases with high frequency of change; e.g., ground truth
  - Create judicious partitions
  - Minimize local virtual time difference between partitions and those objects which use them

MITRE

IMPLEMENTATION WARNING SIGNALS

- High number of messages
  - Bad allocation of objects to processor
- High number of anti-messages
  - Bad allocation of objects to processor
  - Poor clustering
  - Poor entity definition
  - Inherently serial problem
- High number of rollbacks
  - Bad allocations of objects to processor
  - Processor load is not balanced
  - Inherently serial problem

MITRE

TIGHTENING OF ASSUMPTIONS

CONSTRAINTS

- Processor Memory Size
  - Limits the number of old state copies saved, and therefore ease of rollback
  - Limits degree of clustering
  - Bounds amount of stored information and procedures
- Slow Communications
  - Slows simulation
  - Increases frequency and magnitude of rollbacks

SUGGESTED TECHNIQUES

- Paging
- Judicious allocation and collocation of entities
- Judicious handling of auxiliary objects
- Load balancing
- Frequenct updates of GVT

MITRE FINDINGS/OBSERVATIONS

PERFORMANCE

- Performance will depend on
  - good problem decomposition
  - good object, procedure and data assignments among processors
- Testing and iteration required
- Major bottlenecks
  - sequential nature of C^2 processes at higher levels
  - large common data bases

MITRE FINDINGS/OBSERVATIONS

INTEGRATION WITH CURRENT ARCHITECTURES

- Hierarchical concept would be facilitated if FORCEM and CASTFOREM also object-oriented
- (But) flow of data (scenario, results) usually requires messaging anyway

MITRE

FINDINGS/OBSERVATIONS

BETTER C^2 REPRESENTATION

- Object orientation forces designer/programmer to think through C^2 process
- Object orientation provides modularity which eases modification of represented C^2
- Hardware/software mechanism to provide quick enough processing to use object-oriented programming thereby obtaining better C^2 representation

MITRE

FINDINGS/OBSERVATIONS

RESOURCE REQUIREMENTS

- Heavy design requirements
- Heavy modification requirements
- Dedicated system when operational
- User skill level dependent on nature of development model

MITRE

FINDINGS/OBSERVATIONS

EFFECT ON USE OF OBJECT ORIENTED TECHNIQUES

- Design pushes problem decomposition to objects
- Message passing enforced
- Tracing available
- Familiar capabilities not available in prototype (and C)
- Simulation tool package required
  - hierarchy
  - graphics
  - modification

MITRE

CONCLUSIONS

- TW/HC concept worth pursuing
- Two reasons to limit expectations
  - difficulty of problem decomposition
  - tradeoffs of processing and communication
- To take maximum advantage
  - need smart decomposition of problem
  - need smart assignment of objects
  - need smart way to handle common data bases and functions

MITRE

SUMMARY (Cont'd)

- Considerable effort in construction and modification of a command and control model implemented on the TW/HC mechanism can be expected
- A good package of simulation tools is needed

GLOSSARY

AMIP	Army Model Improvement Program
AMMO	Army Model Improvement Program Management Office
BEM	Battlefield Environment Model
C^2	Command and Control
CORDIVEM	Corps and Division Evaluation Model

BIBLIOGRAPHY

1. Adams, Loyce and Crocket, Thomas, "Modeling Algorithm Execution Time on Processor Arrays," Computer, vol. 17, n. 7, July 1984, pp. 38-43.

2. The MITRE Corporation, Functional Area Representation Objectives (FAROs) for the Corps/Division Evaluation Model (CORDIVEM), MTR-82W00175, Rev 1, R. P. Bonasso et al., October 1982.

3. Fahlman, Scott, Geoffrey, Hinton and Sejnowski, Terrance "Massively Parallel Architectures for AI: NETL, THISTLE, and BOLTZMANN machines", Proceedings ofthe National Conference on Artificial Intelligence, 22-26 August 1983, pp. 109-113.

4. Jefferson, David, "Virtual Time," Proceedings of the 1983 International Conference on Parallel Processing, IEEE, 23-26 August 1983, pp. 384-394.

5. Jefferson, David etal., "Implementation of Time Warp on the Caltech Hypercube," Presented at the Society for Computer Simulation Conference on Distribute Simulation, San Diego, California, January 1985.

6. Jefferson, David and Muntro, A.,The Time Warp Mechanism for Database Concurrency Control, University of Southern California Technical Report TR-84-302, Los Angeles, California, January 1984.

7. Jefferson, David and Sowizral, Henry, Fask Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control, A RAND Note, N-1906-AF, The RAND Corporation, Santa Monica, California, June 1983.

8. Jefferson, David and Witowski, Andrew, "An Approach to Performance Analysis of Timestamp-driven Synchronization Mechanisms," University of Southern California, Los Angeles, California, June 1984.

9. The RAND Corporation, Simulating Warfare in the Ross Language, A RAND Note, N-1885-AF, Klahr, Philip et al., September 1982.

10. The MITRE Corporation, The Army Model Improvement - A Technical Assessment of Selected Army Models, MTR82-W-00198, Nugent, Richard O., November 1982.

11. The MITRE Corporation, "MITRE's Battlefield Environment Model," Working Group Presentation, ORSA-TIMS Conference, Nugent, Richard O., November 1984.

BIBLIOGRAPHY (Concluded)

12. The MITRE Corporation, A Preliminary Evaluation of Object-Oriented Programming for Ground Combat Modeling, WPW00407, Nugent, Richard O., September 1983.

13. The RAND Corporation, The Ross Language Manual, A RAND Note, N-1854-AF, McArthur, David and Klahr, Phillip , September 1982.

14. Seitz, Charles, "The Cosmic Cube," Communications of the ACM, vol. 28, n. 1, January 1985, pp. 22-33.

DISTRIBUTION LIST

INTERNAL

A-10 C. A. Zraket

D-14 A. J. Roberts

W-25 S. Bradley
M. G. Walker

W-70 A. W. Bowers
P. G. Freck
R. P. Granato
R. A. Joy
Dr. F. W. Niedenfuhr

W-72 E. J. Boyle
G. S. Bullen
C. W. Sanders

W-73 C. V. Moran
T. H. Nyman

W-74 T. T. Bean
R. P. Bonasso
Dr. J. C. Ellenbogen
R. L. Hegerich
C. Jackson
K. N. Karna
E. H. Kasif
S. J. Laskowski (5)
A. M. Lidy
R. T. Nixon
R. O. Nugent (5)
G. A. Simpson
Dr. L. M. Sokol (5)

W-75 M. Barov
Z. Z. Friedlander
J. Rubin

W-76 C. R. Holt
G. K. Holt
W. E. Zeiner

W-90 J. W. Benoit

EXTERNAL

DARPA
LTC M. Montie (2)
Architect Building
TTO 10th Floor
1400 Wilson Boulevard
Arlington, VA 22209

The Army Model
Management Office
U.S. Army Combined Armys Center
ATTN: ATZL-CAN-DO
COL K. E. Wiersema (5)
Ft. Leavenworth, KS 66027

HQ, Department of the Army
SAUS-OR (Mr. Hollis)
DAMA-ZD (Mr. Woodall)
DACS-DMO (Ms. Langston)
ASA (IL&FM) (Mr. Rosenblum)
Washington D.C. 20310

Jet Propulsion Laboratories
J. Tupman
4800 Oak Grove
Pasadena, CA 91109

Institute for Defense Analysis
Program Analysis Division
Dr. L. B. Anderson
1801 North Beauregard Street
Alexandria, VA 22311

DISTRIBUTION LIST (Continued)

EXTERNAL

U. S. Army Ballistic
Research Laboratory
AMXBR-SECAD
ATTN: S. Wolff
Aberdeen Prov. Grd., MD 21005-5066

Army Library
ATTN: ANR-AL-RS
(Army Studies)
Room 1A518
Pentagon
Washington D.C. 20310

Commander
Defense Technical
Information Center
ATTN: DDA
Cameron Station
Alexandria, VA 22314 (2)

Commandant
U.S. Army Command and
General Staff College
Ft. Leavenworth, KS 66027

Commandant
U.S. Army War College
Carlisle Barracks, PA 17013

Commander
U.S. Army Combined Arms Center
ATZL-CSC-I
ATZL-CAS-W
ATZL-TAC-LO
Ft. Leavenworth, KS 66027

U.S. Army Intelligence Center
ATTN: ATSI-CD-CS
Ft. Huachuca, AZ 85613

Director, U.S. Army Concepts
Analysis Agency
ATTN: Dr. R. B. Modjeski
8120 Woodmont Avenue
Bethesda, MD 20014

Director, U.S. Army
Material Systems
Analysis Activity
Aberdeen Proving Grd., MD 221005

Director, U.S. Army
Research Institute
ATTN: PERI-SZ
Dr. Johnson
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Wilbur Payne
Director TRADOC Operations
Research Activity
White Sands, NM 88002

Deputy Commander
Combined Arms Systems
Analysis Activity
ATTN:ATOR-CAA-DC
ATOR-CAA-DR
Ft. Leavenworth, KS 66027

Commander
U.S. Army Training and
Doctrine Command
ATTN: ATCG-S (Mr. Christman)
Ft. Monroe, VA 23651

Lawrence Livermore Laboratory
Attn: Stan Erickson, L-7
P.O. Box 808
Livermore, CA 94550

EXTERNAL

University of California,
Dept. of Computer Science
Prof. D. Jefferson
3531 Boelter Hall
Los Angeles, CA 90024

The RAND Corporation
Dr. P. Klahr
Dr. L. Joe
Dr. M. Milhalka
1700 Main Street
Santa Monica, CA 90406

Rome Air Development Center
Mr. Andrew Kozak (IRDT)
Dr. Northrop Fowler (COES)
Griffiss AFB, NY 13441

Eaton Corporation
AI Division
Commack Road
Deer Park, NY 11729

BDM
Mr. J. Kenney
1300 N. 17th Street
Suite 950
Arlington, VA 22209

END
DTIC
4-86